Least Authority
PRIVACY MATTERS

Operator AVS + Smart Contracts (Second Review)

Security Audit Report

Aligned Layer

Final Audit Report: 2 September 2025

Table of Contents

Overview

Background

Aligned Layer has requested that Least Authority perform a second security audit of their Operator AVS and smart contracts. Aligned Layer is a verification layer for zero-knowledge proofs using Eigen Layer.

Project Dates

- **November 21, 2024 -** November 29, 2024: Initial Code Review *(Completed)*
- **December 2, 2024:** Delivery of Initial Audit Report *(Completed)*
- **September 1, 2024:** Verification Review *(Completed)*
- **September 2, 2024:** Delivery of Final Audit Report *(Completed)*

Review Team

- Will Sklenars, Security Researcher and Engineer
- Dominic Tarr, Security Researcher and Engineer
- Burak Atasoy, Project Manager
- Jessy Bissal, Technical Editor

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Operator AVS and smart contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in scope for the review:

- Aligned Layer:
  https://github.com/yetanotherco/aligned_layer

Specifically, we examined the Git revision for our initial review:

- `1125be82b7c149bfe27932e6ee3a4c0ff00a1c5b`

For the verification, we examined the Git revision:

- `15c8d13ec156f64cafe030e923ae8c36e94c80d3`

For the review, this repository was cloned for use during the audit and for reference in this report:

- https://github.com/LeastAuthority/Aligned-Layer/tree/audit2

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Website:
  https://alignedlayer.com
- Aligned Layer Documentation:
  https://docs.alignedlayer.com
- `Eigenlayer` Whitepaper:
  https://docs.eigenlayer.xyz/html/EigenLayer_WhitePaper-converted-xodo.html#bookmark34

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are passed correctly to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Our team recently performed a security audit of the Aligned Layer system as part of a multi-firm audit, targeting Aligned Layer version `0.4.0`. In this review, we performed a follow-up audit, focusing on the changes between Aligned Layer version `0.4.0` and `0.11.2`. In the most recent updates, the Aligned Layer team has addressed some of the issues raised by the recent audits. Overall, our team found that version `0.11.2` contains a few new features, and that system reliability and efficiency was improved.

System Design

Since version `0.4.0m`, the Aligned Layer team has taken steps to improve the overall reliability of this system. This includes fixing a memory leak, and an IO-related (in/out) vulnerability raised in our previous audit. The Aligned Layer team has also added retry logic for network communications, and added the functionality for a user to be able to increase the fees allowance for a previously submitted task. This solves the scenario where a user's task is stuck in the task queue because there are other pending tasks with higher fees allowances. To increase efficiency and reduce gas costs, the Aligned Layer team has removed the Merkle root check from the `BatcherPaymentService`. However, due to this, the Batcher currently has excess authority (Issue D).

Our team identified some areas of concern in the Batcher. Currently, the pending task queue is reinitialized to an empty state whenever the Batcher is restarted, for example, due to a new version of the Batcher being deployed (Issue B). The Batcher task queue is also vulnerable to a denial of service (DoS) attack due to the queue growing excessively and consuming all available system memory. This could either occur under normal usage patterns, or become a vector for an attack (Issue A). Our team also identified a

DoS attack scenario where an attacker can replay tasks, consuming CPU cycles on proof validation and ultimately causing system failure.

Code Quality

We performed a manual review of the repositories in scope and found that the code is well organized and generally follows the best practices of the languages used.

Tests

The repositories in scope include some tests that cover the basic functionality of the system; however, there was no test coverage for the Batcher or Aggregator.

Documentation and Code Comments

The high level project documentation provided by the Aligned Layer team was accurate and helpful in describing the intended functionality of the system. Additionally, we found that the code is moderately commented.

Scope

The scope of this review was sufficient, as it included the changes implemented since our last review, in addition to the PR relating to the `eigenlayer-middleware`.

Dependencies

Our team noted that several dependencies are implemented in the codebase; however, we did not identify any security concerns resulting from their unsafe usage.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Batcher Denial of Service (DoS) Attack Due to Task Queue Overflow	Resolved
Issue B: Batcher Downtime Results in Pending Tasks Being Dropped	Planned
Issue C: Replay-Based DoS Attack on Batcher	Resolved
Issue D: Batcher Has Excess Authority	Planned

Issue A: Batcher Denial of Service (DoS) Attack Due to Task Queue Overflow

Location

`Aligned-Layer/tree/audit2/batcherBatcher`

Synopsis

An attacker could submit many tasks to the Batcher in order to fill up the task queue and cause the Batcher to run out of memory. The task queue may also become overinflated if legitimate use overextends the maximum task processing capacity of the Batcher, or if users submit tasks with insufficient fee allowance. The tasks queue could additionally become overinflated from legitimate use.

Impact

The Batcher would crash. Users leveraging the Batcher to submit tasks would not be able to reach the service. Any tasks already in the Batcher queue would be dropped, as the queue is stored in memory.

Preconditions

The attacker must have a positive balance in the `BatcherPaymentService` contract greater than the sum of the `max_fee` property in the tasks they submit. However, the balance would not be depleted if the Batcher crashes before any of the attacker's tasks propagate through the Aligned Layer system, and the funds could be withdrawn after the attack. In addition, the value of the `max_fee` property is set by the attacker, and could be set to be very low. This would require fewer funds, and make it less likely that the attacker's tasks will propagate through the system, as there may be other tasks with a higher `max_fee` value that will be given priority.

Feasibility

The attack is possible with the current version of the system.

Technical Details

The largest item in a submitted task is the proof itself. The attacker would likely choose a proving system that generates large proofs, such as RISC Zero. If the attacker uses `300`Kb RISK Zero proofs, and if the Batcher has 8GB of memory available, the attacker would need to submit approximately `28,000` tasks in order to exhaust the memory.

The task queue may become overinflated if legitimate use overextends the maximum task processing capacity of the Batcher (`1500` tasks per block). Moreover, if users submit tasks with insufficient fee allowance, these tasks will stay in the task queue and may never be removed, taking up memory.

Remediation

We recommend writing tasks on an on-disk data structure first, and then ingesting them into the in-memory queue. The queue system could be extended, such that it has a maximum length so that memory resources are never exhausted. We further recommend adding a minimum value requirement for the user-supplied `max_fee` argument to increase the balance required to leverage the attack. Additionally, rate limiting could also be applied so that an attacker is not able to rapidly submit tasks.

To solve the problem of tasks languishing in the queue because of low fees allowance, the task queue could also have a garbage collection process where tasks are removed from the queue if they have been in the queue for longer than a certain expiry period.

Status

The Aligned Layer team has introduced a configurable `max_size` for the task queue, which defaults to `10,000` items. If the queue is full, any additional items will return an error. Since `28,000` large items were previously estimated to be required to perform the attack, a limit of `10,000` should be sufficient.

Verification

Resolved.

Issue B: Batcher Downtime Results in Pending Tasks Being Dropped

Location

[Aligned-Layer/tree/audit2/batcher]

Synopsis

If the Batcher experiences downtime due to, for example, a software update, any pending tasks in the task queue that have not yet been included in a batch will be dropped.

Impact

Such a pending task will not propagate through the system. A user will not know that their task has been dropped and may assume that it is still in the queue, possibly stuck because of an insufficient `max_fee` value.

Preconditions

For this issue to occur, there should be at least one pending task in the queue, and the Batcher would have to be experiencing downtime. Downtime could be caused by a planned update, a software bug, or a hardware fault.

Feasibility

As software updates and outages are to be expected, this scenario is likely to occur.

Technical Details

The task queue is kept in volatile memory and is not backed up to disk. Therefore, after an application restart, the queue is reinitialized to an empty array.

Mitigation

Users who notice that their task has not propagated through the system and who do not know whether this is because of an outage or a `max_fee` value that is too low can resubmit the task with a higher `max_fee` value. This would mitigate both potential reasons that their task has not been included in a batch.

Remediation

We recommend saving tasks submitted to the Batcher to an on-disk data structure first, before ingesting them by the in-memory task queue. When a task is successfully included in a batch, the task can then be deleted from the on-disk data structure. This would ensure that any tasks residing in the in-memory task queue that have not yet been included in a batch are also backed up on disk, and can be read again when the Batcher comes back online after downtime.

Status

The Aligned Layer team acknowledged the issue and noted that, although the remediation has not been undertaken due to current time constraints, it is planned for a future update.

Verification

Planned.

Issue C: Replay-Based DoS Attack on Batcher

Location

batcher/src/lib.rs:528-773

Synopsis

The Batcher performs several checks before accepting a proposed proof as valid. However, because it executes an expensive check (proof verification) early on in the process, this can be exploited for a DoS attack.

Impact

An attacker could cause the Batcher to become unresponsive.

Preconditions

The attacker requires a proof (preferably one that takes longer to verify) with a valid signature, along with a connection that allows rapid submission to the Batcher.

Feasibility

Straightforward.

Technical Details

Several checks are performed on the incoming `SubmitProofMessage` before it is accepted.
The signature must be valid and the proof length must be acceptable. If preverification is enabled (and in production it must be) the proof is verified, and a number of checks then ensure the user has sufficient balance to cover the proof, after which the nonce is finally checked. If the nonce is greater than expected, the message is dropped. If the nonce is old, it is accepted as an update only if the fee has been increased; otherwise, it is dropped. If the nonce is the expected (correct) nonce, then it is accepted and added to the batch.

An anonymous attacker without balance in the system can use a previously signed `SubmitProofMessage` from another user and submit it to the Batcher. Since it has a valid signature, it will pass the signature check. Similarly, since it has a correct proof, it will pass the proof check (although this is a non-trivial CPU load). If the user has sufficient balance to submit the proof, it will pass the balance checks. Finally, once it reaches the nonce check, since it has an old nonce, it will be discarded.

A message with an old nonce is only accepted if there is a proof with the matching nonce already in the queue. In this case, the new update must have a higher fee. If an attacker replays an old message, it will not change the fee, causing it to be dropped.

Since the proof verification is fairly expensive, an attacker could submit a significant number of proofs, causing the Batcher CPU to become fully occupied. However, this cannot be traced back to them, as the attacker would have replayed someone else's `SubmitProofMessage`.

In the previous audit, our team found that the signature was sent to the `AlignedLayerServiceManager` and verified. Due to this, an attacker could take a valid signature and proof from the public blockchain. In the current audit, this has been removed, causing Issue D, as explained below. Issue D is more critical than Issue C, so when Issue D is remediated, Issue C must be resolved as well.

Remediation

We recommend performing the nonce check before the proof verification. It is good practice to perform the most expensive checks last; however, to protect against replays it is sufficient to perform the nonce check alongside the signature check.

Status

The Aligned Layer team has implemented a process in which the proof is verified last, while less expensive checks, including nonce, signature, and whether the user has sufficient balance are performed first.

Verification

Resolved.

Issue D: Batcher Has Excess Authority

Location

`contracts/src/core/AlignedLayerServiceManager.sol:56-80`

Synopsis

Signature checks were removed from `createNewTask`. Due to this change, the Batcher currently has the power to attribute batches to any Aligned Layer user, charging the user for the batch fees, which are then transferred to the Batcher's account and can be withdrawn by the Batcher.

Impact

The system cannot be considered decentralized, as the Batcher has unnecessary power over user funds.

Preconditions

A user must have a balance in the system.

Feasibility

This issue can occur if the attacker controls the Batcher wallet.

Technical Details

In `pull request #1114`, the signature checks that were previously used in `AlignedLayerServiceManager.createNewTask` were removed to save a significant amount of gas per call. Instead of verifying the user's request to the Batcher with their signature, the Batcher provides a list of accounts to be charged. Without a signature check, there is no way for the service manager contract to verify that the user actually submitted a batch. Consequently, the Batcher can charge any user. Previously, `createNewBatch` was publicly accessible, making the system more decentralized by allowing anyone to run their own Batcher; however, not only does the Batcher currently have a trusted centralized role, but it is also not possible to run an alternative Batcher.

Mitigation

A user cannot prevent the Batcher from stealing from them, but they can reduce their exposure to this risk by not having a large balance within the Aligned Layer system.

Remediation

To have a strongly decentralized system, we recommend implementing a mechanism that checks user signatures to verify that they did submit a proof. Although checking signatures inside the contract is expensive, there are other ways a signature could be verified. One approach would be to include the signatures within the batch and have the operators verify the signatures along with the proofs. The list of signatures should be integrated into the Merkle tree, such that correct signatures are strongly tied into the batch to ensure that users can verify their signature as well. If the user was charged after the batch was verified, it would be possible to reinstate the public Batcher, restoring strong decentralization.

Status

The Aligned Layer team acknowledged the issue and noted that, although the remediation has not been undertaken due to current time constraints, it is planned for a future update.

Verification

Planned.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.